**Amendments to the Claims:**

This listing of claims will replace all prior versions, and listings, of claims in the application:

**Listing of Claims:**

**Claim 1. (Cancelled).**

**Claim 2. (Currently Amended)** A method for managing flow of messages between two software modules, said method comprising:

(a) determining whether a first message can be propagated to or from a first segmented synchronization queue associated with one of the two software modules, by a first thread running on a first processor, while allowing a second thread running on a second processor to propagate a second message between the two software modules; and

(b) propagating the first message to or from the first segmented synchronization queue while allowing the second thread to propagate the second message between the two software modules when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules wherein the first and second threads concurrently propagate respective portions of the first and second messages to or from the first segmented synchronization queue.

**Claims 3 & 4. (Cancelled)**

**Claim 5. (Previously Presented)** A method as recited in claim 2, wherein the method further comprises:

setting a first indicator for the first processor to indicate that the first processor is propagating when said determining (a) determines that the first message can be propagated by

the first thread while allowing the second thread to propagate the second message between the two software modules; and

setting the first indicator for the first processor to indicate that the first processor is not propagating when said propagating (b) has propagated the first message between the two software modules.

**Claim 6. (Previously Presented)** A method as recited in claim 2, wherein said determining (a) comprises:

(a1) determining whether an event is being processed or is pending to be processed; and

(a2) determining whether a thread-count for the first processor is zero, and

wherein said determining (a) determines that the first thread can propagate the first message without blocking the second thread from propagating the second message when said determining (a) determines that no events is being processed or pending and the thread-count for the first processor is zero.

**Claims 7-9. (Cancelled)**

**Claim 10. (Previously Presented)** A method as recited in claim 2, wherein the two software modules are implemented in a stack as STREAMS modules.

**Claim 11. (Currently Amended)** A computer system comprising:

a plurality of processors;

first and second software modules, the second software module having a main queue suitable for storing messages and [[an]] a segmented auxiliary queue suitable for storing messages that are not stored in the main queue;

wherein the segmented auxiliary queue is configured for storing and processing operational messages separately from data messages, said operational messages relating to operational events associated with managing flow of data between the first layer software module and a second layer software module;

and

a propagation controller for propagating messages between the first and the second software modules, the propagation controller operating to enable at least two processors of said plurality of processors to concurrently propagate messages to or from the segmented auxiliary queue of the second software module.

**Claim 12. (Currently Amended)** A computer system as recited in claim 11, wherein the propagation controller comprises:

a thread-count for one of said plurality of processors; and

a queue count for the segmented auxiliary queue.

**Claim 13. (Currently Amended)** A computer system as recited in claim 12,

wherein the segmented auxiliary queue is a segmented synchronization queue and the queue count is a segmented synchronization queue count.

**Claim 14. (Currently Amended)** A computer system as recited in claim 11, wherein the at least two processors of said plurality of processors concurrently propagate a message from the first software module to the segmented auxiliary queue of the second software module.

**15. (Original)** A computer system as recited in claim 11, wherein the at least two processors of said plurality of processors concurrently propagate a message from the first software module to the main queue of the second software module.

**16. (Previously Presented)** A computer system as recited in claim 11, wherein the first and second software modules are implemented in a stack as STREAMS modules.

**Claim 17. (Currently Amended)** A computer readable media including computer program code for managing flow of messages between two software modules, said computer readable media comprising:

computer program code for determining whether a first message can be propagated by a first thread running on a first processor to or from a first segmented synchronization queue of one of the two software modules while allowing a second thread running on a second processor to propagate a second message between the two software modules; and

computer program code for propagating the first message to or from the first segmented synchronization queue while allowing the second thread to propagate the second message between the two software modules when said computer program code for determining determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules wherein the first and second threads concurrently propagate respective portions of the first and second messages to or from the first segmented synchronization queue.

**Claim 18. (Cancelled)**

**Claim 19. (Previously Presented)** A computer readable media as recited in claim 17, wherein the computer readable media further comprises:

computer program code for setting a first indicator for the first processor to indicate that the first processor is propagating when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two layer software modules.

computer program code for setting the first indicator for the first processor to indicate that the first processor is not propagating when said propagating (b) has propagated the first message between the two software modules.

**20. (Original)** A computer readable media as recited in claim 19, wherein the two software modules are implemented in a stack as STREAMS modules.

**Claim 21. (Currently Amended)** A method as recited in claim [[1]] 2, wherein the second thread also propagates the second message to or ~~form~~ from the first segmented synchronization queue.

**Claim 22. (Currently Amended)** A method as recited in claim [[1]] 2, wherein the second thread propagates the second message to or ~~form~~ from one of the first segmented synchronization queue and a second synchronization queue, the second synchronization queue being associated with one of the software modules.

**Claim 23. (Cancelled)**

**Claim 24. (Currently Amended)** A method as recited in claim 2, wherein additional threads concurrently propagate messages to or from the first segmented synchronization queue.

**Claim 25. (New)** A method for managing flow of messages between two software modules, said method comprising:

(a) determining whether a first message can be propagated to or from a first synchronization queue associated with one of the two software modules, by a first thread running

on a first processor, while allowing a second thread running on a second processor to propagate a second message between the two software modules;

and

(b) propagating the first message to or from the first synchronization queue while allowing the second thread to propagate the second message between the two software modules when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules wherein the first and second threads concurrently propagate respective portions of the first and second messages to or from the first synchronization queue; and

wherein said determining (a) comprises:

(a1) determining whether an event is being processed or is pending to be processed; and

(a2) determining whether a thread-count for the first processor is zero, and

wherein said determining (a) determines that the first thread can propagate the first message without blocking the second thread from propagating the second message when said determining (a) determines that no events is being processed or pending and the thread-count for the first processor is zero.